US009354847B2

(12) **United States Patent**
Wolf et al.

(10) **Patent No.: US 9,354,847 B2**
(45) **Date of Patent: May 31, 2016**

(54) **INTERFACE INFRASTRUCTURE FOR A CONTINUATION BASED RUNTIME**

(75) Inventors: **Kenneth D. Wolf**, Seattle, WA (US); **Edmund Samuel Victor Pinto**, Duvall, WA (US); **Robert Brian Schmidt**, Woodinville, WA (US); **Donald F. Box**, Bellevue, WA (US); **Geoffrey M. Kizer**, Seattle, WA (US); **Nathan C. Talbert**, Seattle, WA (US); **Kavita Kamani**, Issaquah, WA (US); **Alberto Arias Maestro**, Seattle, WA (US); **David Robert Cliffe**, Bellevue, WA (US); **Tirunelveli R. Vishwanath**, Redmond, WA (US); **HongMei Ge**, Issaquah, WA (US); **Stephen Jared Maine**, Seattle, WA (US); **Alexander Martin DeJarnatt**, Charlottesville, VA (US)

(73) Assignee: **Microsoft Technology Licensing, LLC**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 377 days.

(21) Appl. No.: **12/345,288**

(22) Filed: **Dec. 29, 2008**

(65) **Prior Publication Data**

US 2010/0169862 A1 Jul. 1, 2010

(51) **Int. Cl.**
G06F 9/44 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 8/313*** (2013.01)

(58) **Field of Classification Search**
CPC G06F 8/20; G06F 8/30; G06F 8/31
USPC 717/117
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,490,097	A	2/1996	Senson	
5,748,962	A *	5/1998	Brechtel et al.	719/328
5,918,226	A	6/1999	Tarumi	
5,960,404	A	9/1999	Chaar	
5,999,910	A	12/1999	Rosenfeld	
5,999,911	A	12/1999	Berg	

(Continued)

FOREIGN PATENT DOCUMENTS

GB	2396928	7/2004
JP	2003-331095	11/2003

(Continued)

OTHER PUBLICATIONS

Bukovics, Pro WF Windows Workflow in .NET 3.5, Jun. 26, 2008, Apress, ISBN 978-1-4302-0975-1; Chapters 1-19, 852 pages.*

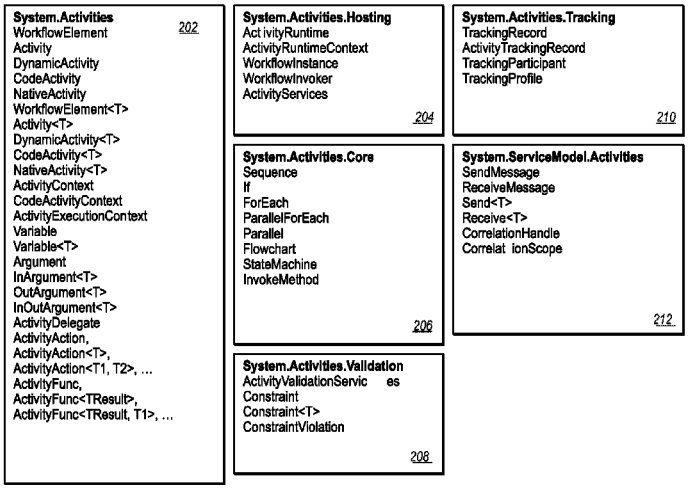
(Continued)

Primary Examiner — Duy Khuong Nguyen
(74) *Attorney, Agent, or Firm* — Ben Tabor; Aaron Hoff; Micky Minhas

(57) **ABSTRACT**

Namespace for continuation-based runtime. Some embodiments described herein are directed to a framework using continuation based runtime namespaces that pertain to an infrastructure for enabling the creation of a wide variety of continuation-based programs that perform a wide-array of tasks. The infrastructure provides a foundation for building continuation-based, declarative applications of various scale and complexity. In some embodiments, the associated application programming interfaces (APIs) are factored into a hierarchy of namespaces in a manner that balances utility, usability, extensibility, and versionability.

12 Claims, 2 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,041,306	A	3/2000	Weiman
6,047,260	A	4/2000	Levinson
6,065,009	A	5/2000	Leymann
6,108,711	A	8/2000	Beck
6,115,640	A	9/2000	Tarumi
6,134,559	A	10/2000	Brumme
6,151,583	A	11/2000	Ohmura
6,253,369	B1	6/2001	Cloud
6,272,672	B1	8/2001	Conway
6,282,531	B1	8/2001	Haughton
6,308,224	B1	10/2001	Leymann
6,339,838	B1	1/2002	Weinman
6,397,191	B1	5/2002	Notani
6,434,568	B1	8/2002	Bowman-Amuah
6,442,528	B1	8/2002	Notani
6,473,794	B1	10/2002	Guheen
6,499,023	B1	12/2002	Dong
6,539,396	B1	3/2003	Bowman
6,578,006	B1	6/2003	Saito
6,675,133	B2	1/2004	Knowles
6,697,784	B2	2/2004	Bacon
6,769,113	B1	7/2004	Bloom
6,772,216	B1	8/2004	Ankireddipally
6,772,407	B1	8/2004	Leymann
6,801,227	B2	10/2004	Bocionek
6,820,118	B1	11/2004	Leymann
6,826,579	B1	11/2004	Leymann
6,839,062	B2	1/2005	Aronson
6,847,974	B2	1/2005	Wachtel
6,854,016	B1	2/2005	Kraenzel
6,877,153	B2	4/2005	Konnersman
6,941,514	B2	9/2005	Bradford
6,968,503	B1	11/2005	Chang
6,983,421	B1	1/2006	Lahti
7,069,536	B2	6/2006	Yaung et al.
7,120,800	B2	10/2006	Ginter
7,240,070	B1	7/2007	Cheng
7,370,335	B1	5/2008	White et al.
7,464,366	B2	12/2008	Shukla et al.
7,698,383	B2	4/2010	Goring
7,827,127	B2	11/2010	Wolf et al.
7,881,233	B2	2/2011	Bieselin
8,862,507	B2	10/2014	Sandhu et al.
2002/0016810	A1	2/2002	Watanabe
2002/0030703	A1	3/2002	Robertson
2002/0032775	A1	3/2002	Venkataramaiah
2002/0038450	A1	3/2002	Matthias
2002/0065701	A1	5/2002	Kim
2002/0147611	A1	10/2002	Greene
2002/0161615	A1	10/2002	Yui
2002/0188597	A1	12/2002	Kern
2003/0004770	A1	1/2003	Miller
2003/0004771	A1	1/2003	Yaung
2003/0018627	A1	1/2003	Turner
2003/0023622	A1	1/2003	Obermeyer
2003/0033191	A1	2/2003	Davies et al.
2003/0055668	A1	3/2003	Amitabh
2003/0078975	A1	4/2003	Ouchi
2003/0090514	A1	5/2003	Cole
2003/0105654	A1	6/2003	Macleod
2003/0135384	A1	7/2003	Nguyen et al.
2003/0181991	A1	9/2003	Chau
2003/0217053	A1	11/2003	Bachman
2003/0233374	A1	12/2003	Spinola et al.
2004/0015841	A1	1/2004	Lepjian
2004/0046789	A1	3/2004	Inanoria
2004/0068728	A1	4/2004	Blevins
2004/0078105	A1	4/2004	Moon
2004/0103014	A1	5/2004	Teegan
2004/0111430	A1	6/2004	Hertling et al.
2004/0117795	A1	6/2004	Wang
2004/0122853	A1	6/2004	Moore
2004/0138939	A1	7/2004	Theiler
2004/0162741	A1	8/2004	Flaxer et al.
2004/0199614	A1	10/2004	Shenfield
2004/0201604	A1	10/2004	Kraenzel
2004/0268338	A1	12/2004	Gurpinar
2005/0005259	A1	1/2005	Avery
2005/0015711	A1	1/2005	Yamamoto
2005/0044173	A1	2/2005	Olander
2005/0066287	A1	3/2005	Tattrie
2005/0096959	A1	5/2005	Kumar et al.
2005/0149908	A1	7/2005	Kilanev
2005/0203757	A1	9/2005	Lei
2005/0234902	A1	10/2005	Meredith
2006/0195347	A1	8/2006	Bultmeyer et al.
2006/0229924	A1	10/2006	Aron et al.
2006/0235964	A1	10/2006	Childress et al.
2006/0294048	A1	12/2006	Shukla et al.
2007/0156485	A1	7/2007	Sanabria
2007/0156486	A1	7/2007	Sanabria
2007/0156888	A1	7/2007	Hilerio
2007/0233969	A1	10/2007	Shukla et al.
2007/0234129	A1	10/2007	Shukla et al.
2007/0239498	A1	10/2007	Shukla et al.
2007/0239499	A1	10/2007	Shukla et al.
2007/0239505	A1	10/2007	Shukla et al.
2007/0245300	A1	10/2007	Chan et al.
2008/0040417	A1	2/2008	Juncker
2008/0114628	A1	5/2008	Johnson
2008/0127156	A1	5/2008	Buza et al.
2008/0167925	A1	7/2008	Mehta et al.
2008/0243524	A1*	10/2008	Agrawal G06Q 10/10 705/1.1
2010/0036859	A1	2/2010	Pinto et al.
2010/0070422	A1*	3/2010	Kikuchi G06F 9/5038 705/301
2010/0306778	A1	12/2010	Wolf et al.
2010/0324948	A1	12/2010	Kumar
2013/0125136	A1	5/2013	Kalra

FOREIGN PATENT DOCUMENTS

JP	2005-506618	3/2005
JP	2005-63253	7/2005
KR	1020010063810	7/2001
WO	WO 02-21314	3/2002
WO	WO 2004-055633	7/2004
WO	WO 2004-059938	7/2004
WO	WO 2004-077262	9/2004

OTHER PUBLICATIONS

Oracle, "Analytic Calculation Engine Metadata Classes", Mar. 2007, pp. 1-70.*

Christopher J.F. Pickett and Clark Verbrugge and Allan Kielstra, libspmt: A Library for Speculative Multithreading, http://www.sable.mcgill.ca/publications/techreports/2007-1/pickett-07-libspmt-TR.pdf, Mar. 12, 2007, 22 Pages.

Alexandr Savinov, Concept as a Generalization of Class and Principles of the Concept-Oriented Programming, http://conceptoriented.com/savinov/publicat/csjm_05.pdf, 2005, 43 Pages.

Raoul A.F. Bhoedjang, Communication Architectures for Parallel-Programming Systems, http://dare.ubvu.vu.nl/bitstream/1871/11711/1/5242.pdf, 2000, 282 Pages.

Peng Li and Steve Zdancewic, A Language-based Approach to Unifying Events and Threads, http://www.cis.upenn.edu/~stevez/papers/LZ06b.pdf, Apr. 2006. 15 Pages.

Peng Li and Steve Zdancewic, A Language-based Approach to Unifying Events and Threads, http://www.cis.upenn.edu/~stevez/papers/LZ06b.pdf, 15 Pages. Apr. 29, 2006.

Muth, et al., "From Centralized Workflow Specification to Distributed Workflow Execution" Journal of intelligent Information Systems, Mar. 1998, vol. 10, No. 2, pp. 159-184, Abstract, 2 pages.

Kappel, et al., "A Framework for Workflow Management Systems Based on Objects, Rules and Roles", 2000, 5 pages.

Moldt, et al., "Pattern Based Workflow Design Using Reference Nets", Lecture Notes in Computer Science, Business Process Management: International Conference, BPM 2003, Jun. 26-27, 2003, pp. 246-260, Abstract, 2 pages.

(56) **References Cited**

OTHER PUBLICATIONS

D. Manolescu, "An Extensible Workflow Architecture with Objects and Patterns", Chapter 4 in Technology of Object-Oriented Languages, Systems, and Architectures, Theo D'Hondt, editor., 2003, 12 pages, Kluwer Academic Publishers.

Leymann, et al., "Workflow-based applications", IBM Systems Journal, 1997, vol. 36, No. I, pp. 102-123.

Huang, et al., "Unified enterprise modeling and integration environment based on Workflow technology", Proceedings of the Third International Conference (ICeCE2003), Oct. 2003, pp. 1000-1003.

Lond, et al., "Accommodating Change in Enterprise Applications", Thesis, IT-University of Copenhagen, 2002, 110 pages.

Vossen, et al., "The WASA2 Object-Oriented Workflow Management System", SIGMOD '99, Philadelphia PA, 1999, pp. 587-589.

Manolescu, et al., "Dynamic Object Model and Adaptive Workflow", OOPSLA Workshop on Metadata and Active Object Models, 1999, 19 pages.

Chen, et al., "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation", HP Labs Technical Report, HPL-1999-136, Oct. 1999, 10 pages, Software Technology Laboratory, Palo Alto, CA.

Ader, et al., "WooRKS, an Object Oriented Workflow System for Offices" IEEE Bulletin of the Technical Committee on Data Engineering, Mar. 1995, 81 pages, vol. 18, No. 1.

Kim, et al., "WW-FLOW: Web based workflow management with runtime encapsulation" IEEE Internet Computing, May-Jun. 2000, pp. 55-64, vol. 4, No. 3.

Miller, et al., "WebWork: METEOR2's Web-based Workflow Management System", Journal of intelligent Information Systems, Special Issue Workflow Management Systems, Mar.-Apr. 1998, pp. 185-215, vol. 10, No. 2.

Windows Workflow Foundation accessible at: http://msdn.microsoft.comlwindowsvistalbuilding!workflow/defaull.aspx, last accessed: Nov. 16, 2005, 4 pages.

International Search Report mailed Jun. 8, 2007, for PCT Application Serial No. PCT/US2006/047220, 8 Pages.

Improving Correctness and Failure Handling in Workflow Management Systems, Kamath, Mohan Umesh, Ph.D., University of Massachusetts Amherst, 1998, 209 Pages; AAT 9841883.

G. Wirtz; M. Weske; H.Giese; The OCoN Approach to Workflow Modeling in Object-Oriented Systems, Information Systems Frontiers, Sep. 2001; 3, 3; ABI/INFORM Global, p. 357-376.

The Workflow Management Coalition—The Workflow Reference Model, Document TC00-1003, Hollingsworth, David, Jan. 19, 1995, pp. 1-55.

Angus, Jeff, "Jet Form's University Deployable Workflow", 1998, Information Week, Iss. 679, p. 104, Pro Quest ID 29161881.

M2 Press Wire, "Team WARE: Team WARE Flow 2.0 Chosen as Product of Choice for Collaboration & Ad-hoc Workflow Apps", Jan. 1998, Coventry, p. 1, ProQuest ID 25682717.

"Conceptual Design and Implementation of a Graphical Workflow-Modeling Editor in the Context of Distributed Groupware-Databases", by Marcus Ott, University of Paderborn Faculty of Business Studies, Germany, May 1994.

"An Evaluation of Methodological Issues in Workflow Management", by Anastasia Sotnikova, Department of Computer Engineering and Information Science and the Institute of Engineering and Science of Bilkent University, Aug. 1998.

"Specification and Implementation of Exceptions in Workflow Management System", by Fabio Casati et al., ACM Transactions on Database System, vol. 24, No. 3, Sep. 1999, pp. 405-451.

"RainMan: A Workflow System for the Internet", IMB T.J. Watson Research Center, Yorktown Heights, NY 1997.

"Window Workflow Foundation Runtime Services: The Persistence Service", http://web.archive.org/web/20051212062613/http://weblogs.asp.net/gsusx/archive/2005/10/05/426699.aspx.

"Action Workflow Enterprise Series 3.0 Process Builder User's Guide", Action Technologies, Inc., 1996.

Dragos A. Manolescu, Workflow enactment with continuation and future objects, In Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pp. 40-51, Seattle, Washington, USA, 2002.

Paul Andrew et al: "Presenting Windows Workflow Foundation, Beta Edition", Sep. 12, 2005, Sams.

Nick Russel et al: "Workflow Data Patterns", Queensland university of Technology Technical Reports, Apr. 2004, pp. 1-75 http://www.workflowpatterns.com/documentation/documents/data_patterns%20BETA%20TR.pdf.

Matthias Kloppmann et al: "WS-BPEL Extension for People", Jul. 2005, pp. 1-18 http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/cfab6fdd-0501-0010-bc82-f5c2414080ed?QuickLink=index&overridelayout=true.

Garcia, Josefina Guerrero, et al., "FlowiXML: A Step Towards Designing Workflow Management Systems", Int J. Web Engineering and Technology, vol. 4, No. 2, 2008, pp. 163-182.

Microsoft, "Streamlined, Customized Workflow Demonstrates Power and Flexibility of Development Platform", Oct. 2007, 12 pages.

Bihler, Pascal, et al., "Supporting Cross-Application Contexts with Dynamic User Interface Fusion", Proceedings of the MoBe Workshop at Informatik 2007, 6 pages.

Hemel, Z., et al., "WebWorkFlow: An Object-Oriented Workflow Modeling Language for Web Applications", 2008, 19 pages.

"Persistent Data Storage with CORBA", by Chris Mayers, ANSA, Poseidon House Castle Park, Cambridge CB3 0RD, UK, Apr. 1996.

"Flexible Persistence Framework for Object-Oriented Middleware", by Mathias Weske and Dominik Kuropka, Hasso Plattner Institute for Software Systems Engineering, Am Luftschiffhafen 1, 14471 Potsdam, Germany. Jun. 25, 2001.

"Openwings Data Services Specification Beta Ver 0.81", General Dynamics Decision Systems, Inc., May 16, 2002.

"A Cooperative Workflow Management System with the Meta-Object Facility", by Le Pallec Xavier and Vantroys Thomas, Laboratoire Trigone—Equipe Noce, Cite Scientifique 59655 Villeneuve D'Ascq Cedex, France, IEEE, 2001.

Ellis, Keddara, Rozenberg; Dynamic Change within Workflow Systems; 1995; ISBN:0-89791-706-5.

Masakazu Sasaki, "Java Development Environment of GUI base to achieves from application development to system integration on BEA WebLogic Platform 8.1J, Try using BEA WebLogic Platform 8.1J", DB Magazine, vol. 13, No. 8, pp. 2-9, Nov. 1, 2003, Shoeisha Co., Ltd., Japan.

Yoshihisa Nakatani, "Java Development Environment of GUI base to Achieves from Application development to system integration on BEA WebLogic Platform 8.1J, Try Using BEA WebLogic Platform 8.1J", DB Magazine, vol. 13, No. 8, pp. 23-28, Nov. 1, 2003, Shoeisha Co., Ltd., Japan.

"Useful! Method for Using an Application Server, Higuchi Laboratory, 15th Installment," Notes/Domino Magazine, vol. 5, No. 8, pp. 136-141, Softbank Publishing Inc., Japan, Aug. 1, 2000.

"New Age of Application Servers, 21st Century for 'Software Industry' Wherein EJB Components Are Commonly Used," Computopia, vol. 35, No. 410, pp. 24-25, Computer Age Co. Ltd., Japan, Nov. 1, 2000.

"Making a J2EE Application a Transparent Service, Scott Dietzen, CTO of BEA, Told its Entire Web Service, BEA WebLogic Server 6.1 & WebLogic Integration," Computopia, vol. 36, No. 420, pp. 84-89, Computer Age Co. Ltd., Japan, Sep. 1, 2001.

Eberle, et al., "Implementation Architectures for Adaptive Workflow Management", In Proceedings of the Second International Conference on Adaptive and Self-Adaptive systems and Applications, 2010, pp. 98-103.

Akram, et al., "Application of Business Process Execution Language to Scientific Workflows", In Proceedings of International Transactions on Systems and Applications, 2006, 14 pages.

"Oracle Workflow", In Proceedings of Oracle Developers Guide, Sep. 2003, 662.

"Softwaremaker", Retrieved on: Oct. 27, 2011, Available at: http://www.softwaremaker.net/blog/CategoryView,category,Windows%2BCommunication%2BFoundation%2B(WCF)%2Baka%2Blndigo.aspx.

"SAP EDI Work Flow Set up Part Three", Retrieved on: Oct. 28, 2011, Available at: http://www.abapprogramming.net/2009_03_01_archive.html.

(56) **References Cited**

OTHER PUBLICATIONS

U.S. Appl. No. 11/321,490, Sep. 22, 2008, Office Action.
U.S. Appl. No. 11/321,490, May 22, 2009, Office Action.
U.S. Appl. No. 11/321,490, Sep. 17, 2009, Office Action.
U.S. Appl. No. 11/321,820, Oct. 14, 2009, Office Action.
U.S. Appl. No. 11/321,777, Dec. 9, 2008, Office Action.
U.S. Appl. No. 11/321,777, Jul. 9, 2009, Office Action.
U.S. Appl. No. 11/321,777, Dec. 11, 2009, Notice of Allowance.
U.S. Appl. No. 11/321,789, Feb. 26, 2010, Office Action.
U.S. Appl. No. 11/321,490, Mar. 19, 2010, Office Action.
U.S. Appl. No. 11/321,820, Mar. 15, 2010, Office Action.
U.S. Appl. No. 11/321,789, Jul. 21, 2010, Office Action.
U.S. Appl. No. 11/321,820, Jul. 20, 2010, Office Action.
U.S. Appl. No. 11/321,820, Nov. 23, 2010, Office Action.
U.S. Appl. No. 11/321,789, Jun. 21, 2011, Office Action.
U.S. Appl. No. 12/487,212, Sep. 26, 2011, Office Action.
U.S. Appl. No. 11/321,789, Dec. 12, 2011, Office Action.
U.S. Appl. No. 11/321,820, Apr. 30, 2012, Office Action.
U.S. Appl. No. 12/487,212, May 2, 2012, Office Action.
U.S. Appl. No. 11/321,490, Sep. 21, 2012, Office Action.
U.S. Appl. No. 11/321,490, Mar. 1, 2013, Office Action.
U.S. Appl. No. 13/296,014, May 1, 2013, Office Action.
Office Action dated Oct. 25, 2013 U.S. Appl. No. 13/296,014.
Office Action dated Nov. 29, 2013 U.S. Appl. No. 11/321,490.
U.S. Appl. No. 14/497,799, filed Sep. 26, 2014, Sanabria et al.
Russell, et al., "Workflow Data Patterns", In Queensland University Of Technology Technical Reports, Apr. 2004, pp. 1-75.
"First Office Action Issued in India Patent Application No. 2201/CHENP/2008", Mailed Date: Feb. 6, 2015, 2 Pages.
U.S. Appl. No. 12/487,212, Mar. 21, 2014, Office Action.
U.S. Appl. No. 11/321,490, May 21, 2014, Notice of Allowance.
U.S. Appl. No. 12/345,288, Sep. 10, 2014, Office Action.
U.S. Appl. No. 13/296,014, Nov. 25, 2014, Office Action.
Office Action dated Jun. 1, 2015 U.S. Appl. No. 13/296,014.
Office Action dated Oct. 15, 2014 U.S. Appl. No. 12/487,212.
Notice of Allowance dated Sep. 25, 2015 U.S. Appl. No. 13/296,014.

* cited by examiner

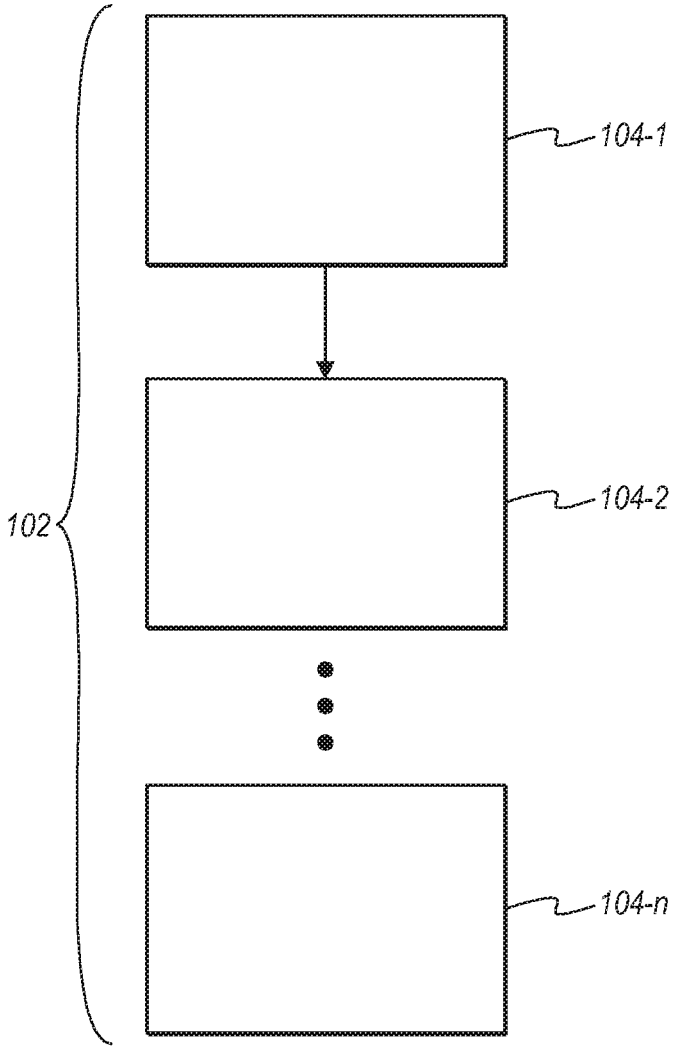


FIG. 1

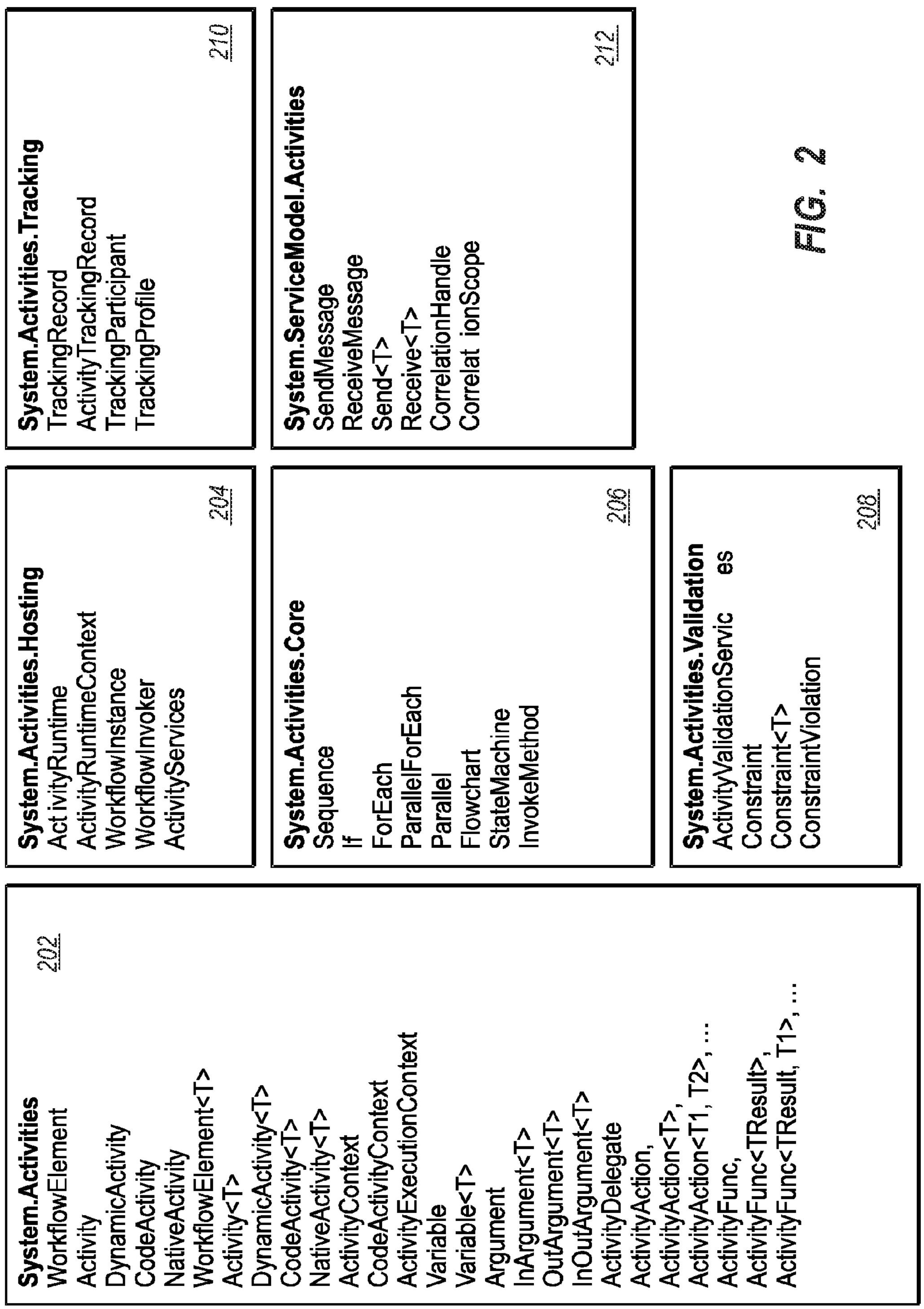

System.Activities
WorkflowElement
Activity
DynamicActivity
CodeActivity
NativeActivity
WorkflowElement<T>
Activity<T>
DynamicActivity<T>
CodeActivity<T>
NativeActivity<T>
ActivityContext
CodeActivityContext
ActivityExecutionContext
Variable
Variable<T>
Argument
InArgument<T>
OutArgument<T>
InOutArgument<T>
ActivityDelegate
ActivityAction,
ActivityAction<T>,
ActivityAction<T1, T2>, ...
ActivityFunc,
ActivityFunc<TResult>,
ActivityFunc<TResult, T1>, ...
202
System.Activities.Hosting
ActivityRuntime
ActivityRuntimeContext
WorkflowInstance
WorkflowInvoker
ActivityServices
204
System.Activities.Core
Sequence
If
ForEach
ParallelForEach
Parallel
Flowchart
StateMachine
InvokeMethod
206
System.Activities.Validation
ActivityValidationServices
Constraint
Constraint<T>
ConstraintViolation
208
System.Activities.Tracking
TrackingRecord
ActivityTrackingRecord
TrackingParticipant
TrackingProfile
210
System.ServiceModel.Activities
SendMessage
ReceiveMessage
Send<T>
Receive<T>
CorrelationHandle
CorrelationScope
212

FIG. 2

INTERFACE INFRASTRUCTURE FOR A CONTINUATION BASED RUNTIME

BACKGROUND

Background and Relevant Art

Computers and computing systems have affected nearly every aspect of modern living. Computers are generally involved in work, recreation, healthcare, transportation, entertainment, household management, etc.

Computer programming can take a number of different forms. For example, computer programmers can develop programs using imperative code. Imperative code is code that directly indicates actions that computer hardware should perform. For example, machine language code may use commands that directly control what is entered into, deleted from, moved to and from, registers and memory locations. Higher level imperative code can be compiled into machine language or binary code to cause the computer hardware to perform desired actions.

Declarative code, in contrast, uses declarative statements to declare the structure of a workflow, where the workflows make general statements about what results should occur from hardware operations rather than defining the specific hardware operations themselves. For example, an XML-based language called XAML is commonly used for declaring the structure of a workflow. Workflows can be used in defining continuation based, declarative applications. However, it can be difficult to build continuation-based, declarative applications of various scale and complexity.

The subject matter claimed herein is not limited to embodiments that solve any disadvantages or that operate only in environments such as those described above. Rather, this background is only provided to illustrate one exemplary technology area where some embodiments described herein may be practiced.

BRIEF SUMMARY

Some embodiments described herein are directed to using a namespace organization to implement a framework for creating and executing continuation-based runtimes. In particular, some embodiments include a computer readable storage media having stored thereon computer executable instructions for implementing a framework for enabling the creation and/or execution of continuation-based declarative applications. The computer readable medium includes a first set of APIs. Each of the APIs is for writing continuation based runtimes. Each of the APIs in the first set of APIs are included in a first common namespace for an activity model of a continuation-based runtime. Each of the APIs in the first set of continuation based runtime APIs are represented with a first common namespace prefix for logical grouping of the plurality of continuation-based runtime APIs.

Other embodiments may organize APIs into other namespaces. For example, a second set of APIs may be organized in a second common namespace where the second set of APIs are used to host, execute, and manage instances of continuation-based, declarative programs. A third set of APIs may be organized in a third common namespace where the third set of APIs are used to implement a base layer of activities that add functionality to the continuation-based runtime and that are used to add concrete behaviors that build on top of a core activity model. A fourth set of APIs may be organized in a fourth common namespace where the fourth set of APIs are used for providing and executing build constraints and policy constraints. Build constraints validate that a given workflow does not violate any runtime or activity assumptions which would most likely cause runtime exceptions. Policy constraints warn of potential issues including one or more of best practice violations, design guideline violations, or company policy violations. A fifth set of APIs may be organized in a fifth common namespace where the fifth set of APIs are used for defining and consuming tracking events. A sixth set of APIs may be organized in a sixth common namespace where the sixth plurality of APIs are used for modeling communication with other systems by sending and receiving messages in and out of a workflow. This set may also include APIs with functionality for representing a continuation-based program as a service.

This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

Additional features and advantages will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the teachings herein. Features and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. Features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

To describe the manner in which the above-recited and other advantages and features can be obtained, a more particular description of the subject matter briefly described above will be rendered by reference to specific embodiments which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments and are not therefore to be considered to be limiting in scope, embodiments will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

FIG. **1** illustrates an example workflow and scheduled activities; and

FIG. **2** illustrates a logical namespace based organization of continuation-based APIs.

DETAILED DESCRIPTION

Some embodiments described herein are directed to a framework using continuation based runtime namespaces that pertain to an infrastructure for enabling the creation of a wide variety of continuation-based programs that perform a wide-array of tasks. The infrastructure provides a foundation for building continuation-based, declarative applications of various scale and complexity. In some embodiments, the associated application programming interfaces (APIs) are factored into a hierarchy of namespaces in a manner that balances utility, usability, extensibility, and versionability.

A continuation based runtime executes activities. An activity represents a unit of executable code including multiple pulses of work. One of the ways an activity can execute multiple pulses of work is through the scheduling of child activities. This composition of activities enables custom control flows that can be implemented through the scheduling of child activities 0, 1 or n times as determined by the composite

activity. An example is illustrated FIG. **1**, which illustrates a workflow **102**, with a number of scheduled activities illustrated generally as **104** and specifically as **104-2**, **104-2**, through **104-*n***, where n is the number of activities in a workflow.

An activity can also setup a resumable continuation in its execution that is resumed by a stimulus external to the runtime. The runtime interprets this external stimulus as another pulse of work to be handled by the activity. Pulses of work are represented internally as continuations that the runtime invokes on activities (thus: continuation based runtime). Activities may also have the following characteristics: They have no process affinity, i.e., they can be paused and resumed in a different process. They have no thread affinity, meaning that different pulses of work can be run on different threads. They can be persisted and rehydrated.

Workflows, like traditional imperative programs, allow one to coordinate work but have some important differences. Workflows can handle long running work by persisting to a durable store, such as a database, when idle and loading again once there is work to do. An instance of a workflow can be modified dynamically while running in the event that new conditions require the workflow to behave differently than it did when it was created. Workflows are a declarative way of writing programs by linking together pre-defined activities rather than an imperative programming model of writing lines of code. Workflows allow one to declare business rules that are separated from code making it easier for one to modify them in the future. Workflows support different styles of systems with sequential and state machine workflows.

An XML-based language called XAML is often used for declaring the structure of a workflow. However, the workflow may also be expressed in other code, such as using any .NET-targeted language (VB.NET, C#, C++/CLI, etc.).

Declarative coding provides developers with the ability to separate the logic of an application from the underlying imperative execution components.

Some embodiments are directed to a continuation based runtime namespace infrastructure for enabling the creation of a wide variety of continuation-based programs that leverage the aforementioned runtime. The infrastructure provides a foundation for building continuation-based, declarative applications of various scale and complexity. The infrastructure or framework provides APIs for continuation-based execution, composite execution, passivatable systems, reactive execution, coordinated sets of inputs, and customized vocabularies

Enhancing features of a continuation-based runtime namespace include a set of activities to perform continuation-based work. Activities come in differing levels of functionality and usability, and include but are not limited to rules-based activities, state machine-based activities, sequential-based activities, database activities, powershell-based activities, web Services-based activities, and Flowchart-based activities.

In some embodiments, the associated APIs are stored on computer readable media (such as those described later herein) and factored logically into a hierarchy of namespaces in a manner that balances utility, usability, extensibility, and versionability. In the following examples, namespaces and classes have been named with specific selected names. However, it should be understood that different namespace names may be selected which implement the same or similar functionality. Notably, APIs may be represented with a common namespace prefix for logical grouping of the APIs depending on what namespace the APIs are organized into. FIG. **2** illustrates the logical organization of APIs within namespaces in a framework. The APIs may be represented, for example, as source code and/or as compiled code. The APIs may be operatable on the .NET platform. The .NET platform, available from Microsoft corporation of Redmond Wash., includes a library of code and a virtual machine that manages the execution of programs written using the code.

Some embodiments include a root namespace for an activity model of the continuation based runtime. In this example, this root namespace is named System.Activities **202**. As noted, this is the root namespace for the activity model of the continuation-based runtime. Specifically, it represents the APIs that are utilized to write a scalable, continuation-based, declarative program. This set of APIs is grouped together in part because it represents a logical, core layer of the activities model and it is possible to write an application using only those APIs in the root namespace and no other continuation-based runtime namespaces. The following illustrates a number of classes (with names selected for readability, but which may be modified in specific embodiments) any or all of which may be organized into the root namespace for an activity model of a continuation based runtime.

One core class of this namespace is illustrated, in this example, as WorkflowElement. The WorkflowElement class represents continuation-based behavior. In particular, instances of the WorkflowElement class include the imperative code in their bodies that is executed by a processor to perform certain functions. This class is used for declaring properties and parameters that are polymorphic across authoring styles.

Hierarchically below the WorkflowElement class, is an Activity class. An Activity is an abstract WorkflowElement (with protected constructor) that is used for declaratively composing behavior into new "types". The execution logic of an Activity is represented by other WorkflowElements. An Activity object may have declared, for example using declarative programming, within its body one or more WorflowElements.

Hierarchically below the WorkflowElement class, is a DynamicActivity class. In the illustrated example, a DynamicActivity is a concrete sealed WorkflowElement (with a public constructor) that is used for declaratively composing behavior "dynamically" using an Activity document object model (DOM). This activity is used by hosters to load activities without requiring new types.

Hierarchically below the WorkflowElement class, is a CodeActivity class. A CodeActivity is an abstract WorkflowElement (with a protected constructor) that is used for writing imperative behavior through an Execute method that has access to data resolution and extensions. CodeActivity is used for writing simple imperative behavior that is semantically equivalent to using Activity with a single-line "InvokeMethod" body (see below). CodeActivity only has an Execute( ) method, and in the present example, it does not have Cancel or Abort functionality.

Hierarchically below the WorkflowElement class, is a NativeActivity class. NativeActivity is an abstract WorkflowElement (with a protected constructor) that is used for writing imperative behavior through an Execute method that has unfettered access to the full breath of the runtime. This includes support for scheduling children, intercepting Cancel( ) and Abort( ) signals, creating continuations, and interacting with execution properties.

Hierarchically below the WorkflowElement class, is a WorkflowElement<T> class. WorkflowElement<T> represents continuation-based "functional" behavior that has a well-known, elevated return value. It is used for declaring

properties and parameters with a return value that are polymorphic across authoring styles.

Hierarchically below the WorkflowElement<T> class is an Activity<T> class. Activity<T> is an abstract WorkflowElement<T> (with protected constructor) that is used for declaratively composing behavior into new "types". The execution logic of an Activity<T> is represented by other WorkflowElements.

Hierarchically below the WorkflowElement<T> class is a DynamicActivity<T> class. DynamicActivity<T> is a concrete sealed WorkflowElement<T> (with a public constructor) that is used for declaratively composing behavior "dynamically" using an Activity DOM. The objects can be used by hosters to load activities without requiring new types.

Hierarchically below the WorkflowElement<T> class is a CodeActivity<T> class. CodeActivity<T> is an abstract WorkflowElement<T> (with a protected constructor) that is used for writing imperative behavior through an Execute method that has access to data resolution and extensions. CodeActivity<T> is used for writing simple imperative behavior that is semantically equivalent to using Activity<T> with a single-line "InvokeMethod" body. In this example CodeActivity<T> only has an Execute( ) method, it does not have Cancel or Abort functionality.

Hierarchically below the WorkflowElement<T> class is a NativeActivity<T> class. NativeActivity<T> is an abstract WorkflowElement<T> (with a protected constructor) that is used for writing imperative behavior through an Execute method that has unfettered access to the full breath of the runtime. This includes support for scheduling children, intercepting Cancel( ) and Abort( ) signals, creating continuations, and interacting with execution properties.

Another base class in the System.Activities **202** namespace is the ActivityContext base class. CodeActivity/CodeActivity<T> and NativeActivity/NativeActivity<T> have contexts passed to them for interfacing with the continuation-based runtime. ActivityContext is the abstract base class for all such contexts. It allows for environment access (i.e. argument/variable resolution) and access to host extensions.

Hierarchically below the ActivityContext class is the CodeActivityContext class. CodeActivityContext is the ActivityContext passed into CodeActivity/CodeActivity<T>.Execute( ). It adds async operation support to ActivityContext.

Hierarchically below the ActivityContext class is the ActivityExecutionContext class. ActivityExecutionContext is the context passed into NativeActivity/NativeActivity<T> methods (Execute, Cancel, and Abort). This class adds support for child management (scheduling, cancelling), setting up no-persist zones, isolation blocks, and bookmarks.

The root namespace System.Activities **202** includes a Variable class. Variable represents the storage of data in a continuation-based program. It has a Type and an (optional) Name.

The root namespace System.Activities **202** includes a Variable<T> class. Variable<T> is a concrete Variable class that strongly represents the type of the Variable through a generic parameter.

The root namespace System.Activities **202** includes an Argument class. Argument is a binding terminal that represents the flow of data for an activity. Arguments declared on an activity definition define the activity's signature. Arguments are bound using expressions that may reference variables as part of activity usage. Arguments have a Type and a Direction (In, Out, or InOut).

The root namespace System.Activities **202** includes an InArgument<T> class. InArgument<T> is a concrete Argument class with a Direction==In, that strongly represents the type of the Argument through a generic parameter.

The root namespace System.Activities **202** includes an OutArgument<T> class. OutArgument<T> is a concrete Argument class with a Direction==Out, that strongly represents the type of the Argument through a generic parameter.

The root namespace System.Activities **202** includes an InOutArgument<T> class. InOutArgument<T> is a concrete Argument class with a Direction==InOut, that strongly represents the type of the Argument through a generic parameter.

The root namespace System.Activities **202** includes an ActivityDelegate class. ActivityDelegate is an abstract base class which represents a snippet of workflow with a specific signature. The signature is a set of parameters and the handler is a WorkflowElement. ActivityDelegate allows an activity author to model parameterized callbacks within the continuation based system.

One or more action classes may be implemented hierarchically below the ActivityDelegate class. Some examples of these include ActivityAction, ActivityAction<T>, ActivityAction<T1, T2>, etc. classes. ActivityAction and related generic types are concrete ActivityDelegates with packaged signatures of zero or more In-parameters.

One or more action classes may be implemented hierarchically below the ActivityDelegate class. These include ActivityFunc, ActivityFunc<TResult>, ActivityFunc<TResult, T1>, etc. ActivityFunc and related generic types are concrete ActivityDelegates with packaged signatures of a single out-parameter and zero or more in-parameters.

Another namespace in the framework is a namespace representing a collection of APIs that are used to host, execute, and manage instances of continuation-based, declarative programs. In this example, the System.Activities.Hosting **204** namespace is the namespace that holds the APIs that are utilized to host, execute and manage instances of continuation-based, declarative programs. This namespace may also include classes implementing the ability to serialize in-flight instances of continuation-based programs. This set of APIs is grouped together because it represents a distinct layer of functionality used to address the problems involved in hosting and executing workflows. Some illustrative classes of this namespace any or all of which may be included in this namespace, illustrated in this example, include the following classes.

The System.Activities.Hosting **204** namespace may include an ActivityRuntime class. ActivityRuntime is a lightweight, per-instance runtime. It can be constructed to create a new workflow instance or to load an existing instance. The APIs that ActivityRuntime exposes are the minimal set of runtime controls from which higher level operations can be implemented. Additionally, this API set represents the entire breadth of functionality of the runtime.

The System.Activities.Hosting **204** namespace may include an ActivityRuntimeContext class. Instances of this class instantiate implementations of the abstract ActivityRuntimeContext and provide the ActivityRuntime access to such host functionality as synchronization, persistence, tracking, and extensions. Additionally, this interface allows the runtime to notify the host of important events such as the transition from running to paused and the occurrence of unhandled exceptions.

The System.Activities.Hosting **204** namespace may include a WorkflowInstance class. WorkflowInstance is a locus of control for workflow instance management. It acts as the thread safe proxy to runtime operations on an instance.

The System.Activities.Hosting **204** namespace may include a WorkflowInvoker class. WorkflowInvoker is a model for directly running an activity as if it were a method call. This is a lightweight, performant, easy to use API for use in scenarios where an activity's execution does not require the control infrastructure provided by WorkflowInstance or ActivityRuntime.

The System.Activities.Hosting **204** namespace may include an ActivityServices class. ActivityServices is the primary entry point for accessing the myriad of hosting services such as Validation, Xaml, Reflection, etc).

The framework may further include a namespace for the base layer of activities that add functionality to the continuation-based runtime. Specifically, this namespace organizes the APIs that are utilized to add concrete behaviors that build on top of the core activity model. In this example, this namespace is represented by System.Activities.Core **206**. As noted, this is a namespace for the base layer of activities that add functionality to the continuation-based runtime. Classes of this namespace may include one or more classes such as are illustrated below.

The System.Activities.Core **206** namespace may include a Sequence class. Sequence is a control-flow activity that is configured with a set of WorkflowElements that it schedules for execution in sequential order (when one completes it will schedule the next one). Sequence is also configured with a set of variables that are used to store state across child activity execution boundaries.

The System.Activities.Core **206** namespace may include an If class. If is a control-flow Activity that is configured with a WorkflowElement-based condition and two WorkflowElements that represent "Then" and "Else" actions of If Then Else logic. Depending on the outcome of the condition, If schedules for execution either the "Then" WorkflowElement or the "Else" WorkflowElement.

The System.Activities.Core **206** namespace may include a ForEach class. ForEach is a control-flow Activity that is configured with a WorkflowElement-based ActivityAction, and an Enumeration to iterate through. ForEach will incrementally go through the enumeration and invoke the configurated "Body" ActivityAction with the current value of the enumeration. When that Action completes, it will get the next value from the enumeration and execute the Action with the new value, etc.

The System.Activities.Core **206** namespace may include a ParallelForEach class. ParallelForEach is similar to ForEach, except that the branch for each value in the enumeration is executed in parallel. It also has an early-completion condition that can be set to break out of the enumeration after a particular condition has been satisfied.

The System.Activities.Core **206** namespace may include a Parallel class. Parallel is a control-flow Activity that is configured with a set of WorkflowElements that it schedules for execution all at once. Parallel is also configured with a set of Variables that are used to store state across child Activity execution boundaries as well as an early-completion condition that can be set to cancel outstanding branches of execution after a particular condition has been satisfied.

The System.Activities.Core **206** namespace may include a Flowchart class. Flowchart is a control-flow Activity that is configured with a set of Nodes that store WorkflowElements for behavior and additional metadata about the next Node that should be scheduled. Flowchart interprets this metadata about nodes into orchestration logic that can follow paths of execution based on steps, decisions, multi-branch switch statements, and more. Flowchart is also configured with a set of Variables that are used to store state across child Activity execution boundaries.

The System.Activities.Core **206** namespace may include a StateMachine class. StateMachine is a control-flow Activity that is configured with a set of States and Transitions. StateMachine interprets this metadata about nodes into orchestration logic that follows the formal models of state machines in executing a reactive system. Events are modeled as WorkflowElements and Conditions control which transition will be followed to enter the next State of processing. StateMachine is also configured with a set of Variables that are used to store state across child Activity execution boundaries.

The System.Activities.Core **206** namespace may include an InvokeMethod class. InvokeMethod is a workflow element which facilitates method calls. InvokeMethod is used for interfacing with common language runtime (CLR) methods and allows users the ability to call methods against objects, pass in parameters, get the return value, specify types for generic methods, and specify if the method is synchronous or asynchronous.

Another namespace that may be included in the framework is a namespace that holds APIs used for providing and executing build constraints and policy constraints. Build constraints validate that a given workflow does not violate any runtime or activity assumptions which would most likely cause runtime exceptions. Policy constraints, on the other hand, warn of potential issues including best practice violations, design guideline violations, company policy violations, and miscellaneous common mistakes. Using these two mechanisms, the APIs in this namespace are used to provide assurances that a validated workflow will execute as the developer intended. In this example, this namespace is referred to as the System.Activities.Validation **208** namespace. The following illustrates a number of classes, any or all of which, may be included in this namespace.

The System.Activities.Validation **208** namespace may include an ActivityValidationServices class. ActivityValidationServices is the mechanism by which workflows can be validated against a set of constraints. As a result of this validation process, a set of violations may be produced.

The System.Activities.Validation **208** namespace may include a Constraint class. The Constraint base class provides the interface between ActivityValidationServices component and the validation constraint that the user wants to implement.

The System.Activities.Validation **208** namespace may include a Constraint<T> class. Constraint<T> provides strongly typed, ActivityAction based sugar on the base class to enable a simpler experience for the constraint author. The ToValidate object from the base Constraint is cast to T and provided as the Body's argument allowing the Body to deal in terms of a strongly typed Variable<T>. This makes authoring easy in both XAML and imperative code.

The System.Activities.Validation **208** namespace may include a ConstraintViolation class. A ConstraintViolation contains the message and error level information of a constraint failure. Note that a single constraint can generate zero or more ConstraintViolations in a single execution.

The framework may include a namespace that holds APIs used for defining and consuming tracking events. The Workflow runtime produces a stream of events that follow its execution. Distinct from the definition of the Workflow, a user may add consumers of these events. In this example, this namespace is referred to as System.Activities.Tracking **210**. This namespace may include a number of classes such as one or more of the following.

The System.Activities.Tracking **210** namespace may include a TrackingRecord class. This is the base class that represents a record containing information about the runtime execution process of a continuation-based program. It defines a common set of fields such as TraceLevel and record number.

The System.Activities.Tracking **210** namespace may include an ActivityTrackingRecord class. This is the base class that represents a record associated with an Activity. It contains the state of the Activity when that record was emitted, the Id of the activity, and values extracted from the running instance at the time the tracking record was emitted.

The System.Activities.Tracking **210** namespace may include a TrackingParticipant class. This is the base class for a consumer of tracking records. A TrackingParticipant can enlist in the Workflow transaction through the PersistenceParticipant mechanism.

The System.Activities.Tracking **210** namespace may include a TrackingProfile class. A TrackingProfile is a declarative definition of filters against event type and the data that should to be queried from the workflow instance (e.g. value of a variable) for a given event. TrackingProfiles can apply to multiple different Workflows and can be associated with multiple TrackingParticipants. TrackingProfiles, their management and their storage are reusable by the authors of TrackingParticipants.

The framework may include a namespace that holds the APIs used for modeling communication with other systems by sending and receiving messages in and out of a workflow. It also includes the APIs with functionality for representing a continuation-based program as a service. In this example, this namespace is referred to as the System.ServiceModel.Activities **212** namespace. Example classes, any or all of which may be included in this namespace, are enumerated below.

The System.ServiceModel.Activities **212** namespace may include a SendMessage class. SendMessage allows users to send data out of a workflow in the form of a message.

The System.ServiceModel.Activities **212** namespace may include a ReceiveMessage class. ReceiveMessage allows users to wait for external stimuli in a continuation-friendly manner by exposing a web service endpoint to receive Messages.

The System.ServiceModel.Activities **212** namespace may include a Send<T> class. Send<T> provides a friendly programming model for sending data that is either a DataContract-based payload, XmlSerializable-based payload, XElement, Stream, or Message.

The System.ServiceModel.Activities **212** namespace may include a Receive<T> class. Receive<T> provides a friendly programming model for receiving data that is either a DataContract-based payload, XmlSerializable-based payload, XElement, Stream, or Message.

The System.ServiceModel.Activities **212** namespace may include a CorrelationHandle class. In workflows, application protocols are decomposed into one-way message exchanges that use a handle based model for relating messaging activities regardless of the underlying correlation mechanism. CorrelationHandle is the name given to the type of handles that facilitate correlation.

The System.ServiceModel.Activities **212** namespace may include a CorrelationScope class. CorrelationScope is used to provide an implicit CorrelationHandle that can be used by Send and Receive activities to loosen the requirements for the user on providing correlation bindings at each individual activity.

Embodiments herein may comprise a special purpose or general-purpose computer including various computer hardware, as discussed in greater detail below.

Embodiments may also include computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any available media that can be accessed by a general purpose or special purpose computer. By way of example, and not limitation, such computer-readable media can comprise RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to carry or store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a computer-readable medium. Thus, any such connection is properly termed a computer-readable medium. Combinations of the above should also be included within the scope of computer-readable media.

Computer-executable instructions comprise, for example, instructions and data which cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A computer readable storage medium, the computer readable storage medium comprising at least one of RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, the computer readable storage medium having stored thereon computer executable instructions for implementing a framework for enabling the creation or execution of continuation-based declarative applications, the computer readable medium comprising computer readable instructions that when executed by one or more processors cause the following to be implemented:

a first plurality of application programming interfaces (APIs), using the processor, configured for writing continuation based runtimes, each of the first plurality of APIs included in a first common namespace for an activity model of a continuation-based. runtime, and each of the first plurality of APIs represented with a common namespace prefix for logical grouping of the first plurality of continuation-based runtime APIs, the first plurality of APIs comprising:

a first API class that represents continuation-based functional behavior, wherein instances of the first API class include imperative code in their bodies that is executed by a processor to perform functions, and wherein the first API class is polymorphic; and

a second API class used to store data in the continuation-based runtime, the second API class including a type and a name, and

a plurality of argument classes, wherein each of the argument classes is a binding terminal that represents the flow of data for instances of the activity class, the plurality of argument classes comprising:

an inargument class that has a direction of In;

an outargument class that has a direction of Out; and

an inoutargument class that has a direction of InOut;

a second plurality of APIs organized in a second common namespace and that are represented with a second common namespace prefix for logical grouping, wherein the second plurality of APIs are used to host, execute, and manage instances of continuation-based, declarative programs;

a third plurality of APIs organized in a third common namespace and that are represented with a third common namespace prefix for logical grouping, wherein the third plurality of APIs are used to implement a base layer of activities that add functionality to the continuation-based runtime and that are used to add concrete behaviors that build on top of a core activity model;

a fourth plurality of APIs organized in a fourth common namespace and that are represented with a fourth common namespace prefix for logical grouping, wherein the fourth plurality of APIs are used for providing and executing build constraints and policy constraints, wherein build constraints validate that a given workflow does not violate any runtime or activity assumptions which would most likely cause runtime exceptions and wherein policy constraints warn of potential issues including one or more of best practice violations, design guideline violations, or company policy violations;

a fifth plurality of APIs organized in a fifth common namespace and that are represented with a fifth common namespace prefix for logical grouping, wherein the fifth plurality of APIs are used for defining and consuming tracking events; and

a sixth plurality of APIs organized in a sixth common namespace and that are represented with a sixth common namespace prefix for logical grouping, wherein the sixth plurality of APIs are used for modeling communication with other systems by sending and receiving messages in and out of a workflow, and for representing a continuation-based program as a service.

2. The computer readable medium of claim **1**, wherein the first plurality of APIs comprises:

a second activity class used for declaratively composing behavior dynamically using an Activity document object model;

a code activity class used for writing imperative behavior through an Execute method that has access to data resolution and extensions;

a native activity class used for writing imperative behavior through an Execute method that has unfettered access to the full breath of the runtime;

a workflow element class used for implementing a continuation-based functional behavior that has a well-known, elevated return value;

an activity context class used for allowing for environment access;

a code activity context class that adds a sync operation support to the API used for allowing environment access;

an activity execution context class that adds support for child management, setting up no-persist zones, isolation blocks, and bookmarks;

a variable class used for the storage of data in a continuation-based program;

an argument class used for implementing a binding terminal that represents the flow of data for an activity; and

an activity delegate class used for implementing a snippet of workflow with a specific signature.

3. The computer readable medium of claim **1**, wherein the first plurality of APIs comprises:

a first API class that represents continuation-based behavior, wherein instances of the class include imperative code in their bodies that is executed by a processor to perform functions; and

a second API class that is used for declaratively composing continuation-based behaviors of the first class into new types by declaring one or more instances of the first API class.

4. The computer readable medium of claim **3**, wherein the first plurality of APIs further comprises:

an API used for declaratively composing behavior dynamically using an Activity document object model;

an API used for writing imperative behavior through an Execute method that has access to data resolution and extensions;

an API used for writing imperative behavior through an Execute method that has unfettered access to the full breath of the runtime;

an API used for implementing a continuation-based functional behavior that has a well-known, elevated return value;

an API used for allowing for environment access;

an API that adds async operation support to the API used for allowing environment access;

an API that adds support for child management, setting up no-persist zones, isolation blocks, and bookmarks;

an API used for the storage of data in a continuation-based program;

an API used for implementing a binding terminal that represents the flow of data for an activity; and

an API used for implementing a snippet of workflow with a specific signature.

5. The computer readable medium of claim **1**, wherein the first plurality of APIs are represented as source code.

6. The computer readable medium of claim **1**, wherein the first plurality of APIs represented as compiled code.

7. The computer readable medium of claim **1**, wherein the second plurality of APIs comprises:

a runtime class used to create a new workflow instance or to load an existing instance;

a runtime context class used to provide the runtime class an existing instance to host functionality including synchronization, persistence, tracking, and extensions;

a workflow instance class used for implementing a locus of control for workflow instance management;

a workflow invoker class used for directly running an activity as if it were a method call; and

a services class used for implementing a primary entry point for accessing a plurality of hosting services.

8. The computer readable medium of claim **1**, wherein the third plurality of APIs comprises:

sequence class used for scheduling instances of the first API class for execution in sequential order;

an if class that is used to schedule one of two workflow elements for execution based on conditions a for each class used for incrementally going through an enumera-

tion and invoking a configurated action class with the current value of the enumeration;

a parallel for each class used for enumerating parallel execution of action classes;

a parallel class used for a control-flow Activity that is configured with a set of instances of the first API class that it schedules for execution all at once;

a flowchart class used for a control-flow Activity that is configured with a set of Nodes that store instances of the first API class for behavior and additional metadata about the next Node that should be scheduled;

a state machine class that is configured with a set of States and Transitions and that interprets this metadata about nodes into orchestration logic that follows the formal models of state machines in executing a reactive system; and

an invoke method class used for facilitating method calls.

9. The computer readable medium of claim **1**, wherein the fourth plurality of APIs comprises:

a validation class used for implementing a mechanism by which workflows can be validated against a set of constraints;

a constraint class used for providing an interface between the API used for implementing a mechanism by which workflows can be validated against a set of constraints and the validation constraint that a user wants to implement;

a second constraint class used for providing strongly typed, action class based sugar on the base class to enable a simpler experience for the constraint author; and

a violation class used for containing the message and error level information f a constraint failure.

10. The computer readable medium of claim **1**, wherein the fifth plurality of APIs comprises:

a tracking record class used for containing information about execution process of a continuation-based program;

activity tracking record class used for that contains the state of an activity when a record is emitted, the Id of the activity, and values extracted from the running instance at the time the tracking record was emitted;

a tracking participant class used for enlisting in a workflow transaction; and

a tracking profile class used for implementing declarative definitions of filters against event type and the data that should to be queried from the workflow instance for a given event.

11. The computer readable medium of claim **1**, wherein the sixth plurality of APIs further include APIs with functionality for representing a continuation-based program as a service.

12. The computer readable medium of claim **1**, wherein the sixth plurality of APIs comprises:

a send message class used for allowing users to send data out of a workflow in the form of a message;

a receive message class used for allowing users to wait for external stimuli in a continuation-friendly manner by exposing a web service endpoint to receive messages;

a second send message class used for providing a friendly programming model for sending data that is either a DataContract-based payload, XmlSerializable-based payload, XElement, Stream, or Message;

a second receive message class used for providing a friendly programming model for receiving data that is either a DataContract-based payload, XmlSerializable-based payload, XElement, Stream, or Message;

a correlation handle class used for implementing handles that facilitate correlation; and

a correlation scope class used to provide an implicit handle that can be used by Send and Receive activities to loosen the requirements for the user on providing correlation bindings at each individual activity. of workflow with a specific signature.

* * * * *